第２章

LSI開発の基礎知識

LSIの開発フローと専門用語を理解する

古川 寛

本章では，これからLSI開発に従事しようとする人のために，LSI開発の一連の流れを解説する．特に，設計と検証に重点を置く．LSI開発の世界では，電子工学出身者であっても聞いたことのない専門用語が多く存在するので，本章ではLSI工程に合わせて，言葉の解説も行う．（編集部）

LSI（large scale integrated circuit）の設計では，多くの専門用語が使われます．新入社員はもちろん，そうでなくとも自身の担当分野以外の用語は不案内であることが多く，共通の言葉で話ができなくなっています．

本章では，LSI開発で用いられる専門用語についても，できるだけ説明します．特に詳しく解説している言葉を**図1**

Keyword

区分	用語	ページ
A～Z	ASIC（application specific integrated circuit）	p.51
	ASSP（application specific standard product）	p.51
	ATPG（automatic test pattern generation）	p.59
	BIST（built in self test）	p.59
	CPF（Common Power Format）	p.60
	CTS（clock tree synthesis）	p.57
	C合成	p.52
	DFM（design for manufacturing）	p.58
	DFT（design for testability）	p.58
	DRC（design rule check）	p.58
	EDIF（electronic design interchange format）	p.53
	ESL（electronic system level）	p.52
	FPGA（field programmable gate array）	p.51
	IDM（integrated device manufacturer）	p.51
	LVS（layout versus schecmatic）	p.58
	OPC（optical proximity correction）	p.58
	RTL（register transfer level）	p.52
	SOC（system on a chip）	p.51
	STA（static timing analysis）	p.55
	TDL（timing driven layout）	p.57
	UPF（Unified Power Format）	p.60
	Verilog HDL	p.52
	VHDL（VHSIC hardware description language）	p.52
あ行	アサーション	p.53
	圧縮パターン・テスト	p.59
か行	機能検証	p.53
	機能設計	p.51
	クロック・ツリー合成	p.57
	形式的検証	p.55
	ゲーテッド・クロック	p.54
	ゲート・アレイ	p.51
	ゲート・シミュレーション	p.55
	ゲート・レベル	p.52
	ゲート遅延	p.54
	検証	p.51
さ行	システムLSI	p.51
	シリコン・ファウンドリ	p.51
	スキャン・インサーション	p.58
	スキャン回路挿入	p.58
	スタティック手法	p.55
	ストラクチャードASIC	p.51
	制約	p.54
	制約付きランダム・テストベンチ	p.53
	設計	p.51
	セットアップ検証	p.56
	セットアップ時間	p.56
	セルベースIC	p.51
た行	ダイナミック手法	p.55
	タイミング・ドリブン・レイアウト	p.57
	タイミング検証	p.55
	タイミング収束	p.58
	タイミング例外	p.56
	ツール・チェーン	p.50
	テクノロジ・マッピング	p.54
	テスト容易化設計	p.58
	同期設計	p.53
	同期リセット	p.53
な行	ネットリスト・レベル	p.52
は行	配線遅延	p.54
	配置配線	p.56
	バック・アノテーション	p.55
	バックエンド	p.50
	パワー検証	p.60
	パワー合成	p.54
	非同期設計	p.53
	非同期リセット	p.53
	ビヘイビア・レベル	p.52
	ファブレス	p.51
	フォーマル・ベリフィケーション	p.55
	フォルス・パス	p.56
	フォワード・アノテーション	p.55
	フロアプラン	p.57
	フロントエンド	p.50
	ホールド検証	p.56
	ホールド時間	p.56
ま行	マルチサイクル・パス	p.56
ら行	リーク電流	p.60
	レイアウト	p.56
	レイアウト検証	p.58
	論理合成	p.54
	論理最適化	p.54
	論理シミュレーション	p.53
	論理等価性検証	p.55

図1　本稿で詳しく解説している専門用語

本文中では，見出しやかぎカッコで示している．

に示します．なお，同じ言葉であっても解釈が異なる場合もありますが，すべてを説明できないためご容赦ください．

1 LSI開発の準備と全体の流れ

最近では，個々の分野の技術レベルが高くなり，LSIの大規模化による負担が増加しています．このため，開発工程ごとに人を割り振る分業制が一般的です．そのせいか，全体像が分からないだけでなく，担当以外の工程に疎いエンジニアが多いようです．

エンジニアとしてのスキルを上げ，良い製品を開発するためには，全体を知ることは大切です．新人であっても，自分の分野とインターフェースする前後の工程には，興味を持ってほしいものです．

● ツール・チェーン

LSIの開発を始めるにあたり，まず最初に**図2**のようなフローを作成します．

フローの作成では，やるべきことをリストアップし，それらを時系列に並べます．そうすることで，必要十分で漏れがないかを確認することができます．実際には使用するEDA（electornic design automation；設計自動化）ツール注1の名前をベースに作成するのが一般的です．このため，「ツール・チェーン」と呼ばれることもあります．

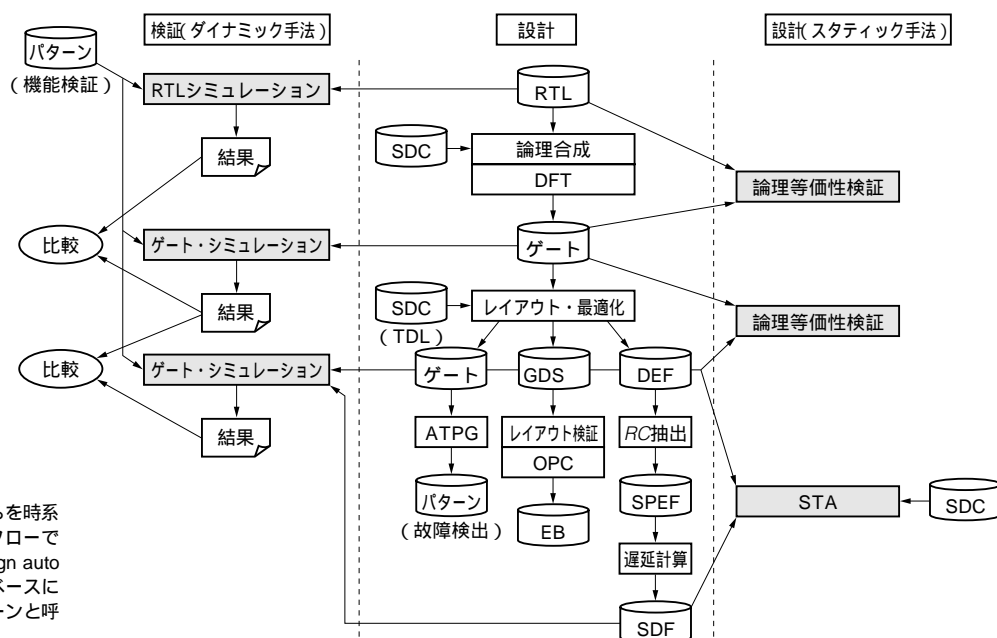
ツール・チェーンの中で一番大事なのが，ツール間のインターフェースです．LSI開発ではさまざまなEDAツールを利用します．いくら個々のツールが良くても，ツール同士でデータのやり取りができないと，活用できません．ツール間でデータをインターフェースするためのファイル・フォーマットも，多くの種類があります．これらを確認しておく必要があります．

LSI開発における最近の話題としては，より上流のESL（electronic system level）と，より下流のDFM（design for manufacturing；製造容易化設計）/DFY（design for yield；歩留まり設計）が挙げられます．**図2**で示した従来の開発フローに対して，上下方向への拡大が進んでいます．まだ広く普及している段階ではないため，図では割愛しました．

上流方向に拡張が進むのは，大規模なLSIを開発するにあたり，設計生産性の向上が求められているためです．下流方向に拡張が進むのは，LSIの製造プロセスの微細化に伴う製造上の問題を解決するためです．

● フロントエンドとバックエンド

ツール・チェーンの時間軸に注目すると，前半を「フロントエンド」，後半を「バックエンド」と呼びます．境界があいまいで，論理合成やDFT（design for testability；テスト容易化設計）をどちらに含めるかはケース・バイ・ケースのようです．少なくとも，機能設計と機能検証はフロントエンド，レイアウト（配置配線）はバックエンドになります．

図2 ツール・チェーン
やるべきことをリストアップし，それらを時系列に並べたもの．LSI開発の一般的なフローである．使用するEDA（electornic design automation；設計自動化）ツールの名前をベースに作成するのが一般的で，ツール・チェーンと呼ばれる．

フロントエンドの山場は，機能を作り上げることです．また，バックエンドの山場は，タイミングを収束させることです．いずれも開発するLSIの仕様として求められています．

● 設計と検証

開発工程は，「設計（design）」と「検証（verification）」に大別できます．EDAツール[注1]で，プラットフォームとして明確に分けられるのが一般的です．

設計は，仕様通りに作ることです．検証は，ほんとうに仕様通りに作られているかどうかを確認することです．人だけでなくツールにもバグはつきものなので，設計した後に必ず検証を行う必要があります．なお，設計は「インプリ」と言う人もいるようです（implementationの省略）．

検証においては，「ダイナミック手法」と「スタティック手法」の2種類に分けられます．基本的にはいずれか一つを行うことになります．最近ではスタティック手法をとる場合が増えています．

● LSIの種類

LSIは，製造方法や流通方法によってさまざまな種類に分類できます．

(1) ASIC（application specific integrated circuit；特定用途向け集積回路）

決まった用途でのみ利用することを目的としたLSIです．機器設計者が自ら利用するために設計したり，設計会社が顧客から依頼によって開発します．製造は半導体メーカが行います．

(2) ASSP（application specific standard product；特定用途向け標準IC）

ASICと同様に用途が決まったLSIですが，半導体メーカが製造し，顧客を特定せずに流通させるものです．

(3) セルベースIC（Cell-based integrated circuit）

半導体メーカがあらかじめ用意した機能ブロック（スタンダード・セル）を利用し，所望の機能が得られるように配置配線して製造するLSIです．スタンダード・セルには，論理ゲート相当の小規模なもの（プリミティブ・セル）から，マイクロプロセッサなどのような大規模なもの（メガセル）までさまざまなものがあります．

注1：昔はCAD（computer aided design；コンピュータによる設計支援）ツールと言われていた．

(4) ゲート・アレイ（gate array）

あらかじめプリミティブ・セルが敷き詰められた構造を採り，配線のみをカスタマイズすることで所望の機能を実現するLSIです．半導体メーカが製造します．

(5) ストラクチャードASIC（structured ASIC）

ゲート・アレイと同様に，配線のみをカスタマイズすることで所望の機能を実現するLSIです．セルとして，比較的大規模な機能ブロックやメモリがあらかじめ用意されている点が異なります．

(6) FPGA（field programmable gate array）

プログラム可能な論理ブロックが敷き詰められた構造を採り，プログラム可能な配線を使って所望の機能を実現できるLSIです．ユーザが手元でカスタマイズできます．論理ブロックの構造や規模が異なるため，CPLD（complex programmable logic device）やSPLD（simple programmable logic device）と呼ばれるLSIもあります．ユーザが手元でカスタマイズできるLSIを総称して，PLD（programmable logic device）と言います．

(7) システムLSIとSOC（system on a chip）

これまでの限られた機能しか備えてなかったLSIに対して，従来のシステム（ボードや装置）に相当する多くの機能を備えている大規模LSIのことをシステムLSIまたはSOC（system on a chip）と言います．IP（intellectual property；知的財産）コアを組み合わせて実現します．

● 半導体メーカの種類

LSI開発にかかわる企業には，さまざまな形態があります．

(1) IDM（integrated device manufacturer）

設計から製造までを行っている垂直統合型の企業を指します．

(2) シリコン・ファウンドリ（silicon foundry）

半導体の製造のみを行う企業を指します．単に「ファウンドリ」と呼ばれることもあります．

(3) ファブレス（fabless）

製造工場を持たない半導体メーカを指します．

2 機能設計

機能設計を始めるに当たり，大きく三つのレベルがあります（**図3**）．レベルの違いは，機能を記述するときの抽象

抽象度

ESL/ビヘイビア・レベル

RTL/HDLレベル

ゲート・レベル/ネットリスト・レベル

図3 設計の抽象度
機能設計では三つのレベルがある．

表1 LSI開発で使われる設計・検証言語

分　野	言語名	標準化
設計言語	SystemC	IEEE 1666
	Verilog HDL (1995/2001)	IEEE 1364
	VHDL (87/93)	IEEE 1076
	SystemVerlog	IEEE 1800
検証（アサーション）言語	SVA (SystemVerilog Assertion)	IEEE 1800
	PSL (Property Specification Language)	IEEE 1850
テストベンチ言語	SVTB (SystemVerilog Testbench)	IEEE 1800
	e	IEEE 1647

度です．RTLで記述してもツールを使ってゲート・レベルに変換するし，ESLで記述してもRTLやゲート・レベルにツールで変換する必要があります．なぜなら，後工程で必要なのはゲート・レベルだからです．

現在はRTL（resister transfer level）が主流です．近い将来，ESLが普及しそうです．かつてはゲート・レベルでした．

抽象度を上げることで，少ない記述で多くの機能を実現できるようになります．このため，設計生産性の向上が期待できます．

● ESL/ビヘイビア・レベルとC合成ツール

最近では，C言語を使った設計を意味する場合が多いようです．抽象度が高いため記述量が減り，それによりバグも減らせることで，設計生産性が上がります．機能検証でも，シミュレーションを高速に実行できます．

現在問題になっているのは，後工程のツールとのインターフェースです．機能設計・機能検証したC言語を，どうやってLSI化するかです．

C言語からRTLへ変換するためには「C合成ツール」が必要になります．最近になって，ようやく実用に耐えうるC合成ツールが発売され始めました．ただし，純粋なC言語そのものが入力できるものはありません．C言語を独自拡張したり，記述に制限が与えられています．LSI開発のためのC言語としては，IEEEで標準化されている「SystemC」があります（**表1**）．

C言語とRTLのコードとの論理等価性の問題もあります．現在では同じテスト・パターンを流して結果を比較するという，間接的な確認しかできません．最近ようやくC言語とRTLの論理等価性検証ツールも出始めましたが，まだ制限も多いようです．

また，C言語で設計するエンジニアの側にも問題があります．ソフトウェア・エンジニアが担当する場合は，LSI設計の知識が不足しがちです．ハードウェア・エンジニアは，C言語に不慣れな場合が多いようです．

● RTL（register transfer level）

現在，HDL（hardware description language；ハードウェア記述言語）による記述の主流がRTLです．このため，このレベルをHDLと呼ぶ人もいます．

HDLとしては，Verilog HDLとVHDL（VHSIC hardware description language）という2種類の言語が業界標準になっています．

現在主流の言語なので，いずれかが書けないと機能設計できないことになります．どちらが主流の言語かと言われると，筆者の周りではVerilog HDLが多いようです．

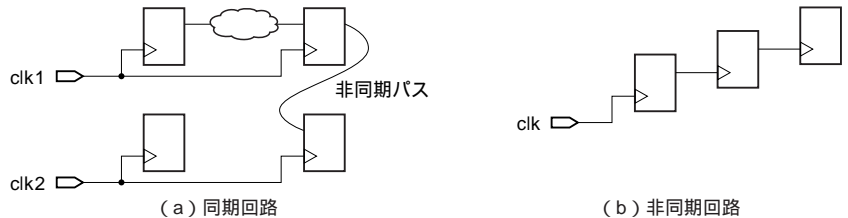
Verilog HDLには，IEEE 1364-1995と，その発展形のIEEE 1364-2001が使われています．ほとんどのツールはすでに2001に対応済みで，さらなる発展形としてSystem Verilogがありますが，ツールは順次対応中の段階です．このため，設計言語としてはまだほとんど使われてないようです．

VHDLは，IEEE 1076-1987とその発展形のIEEE 1076-1993が使われています．いずれもほとんどのツールが対応済みです．

● ゲート・レベル/ネットリスト・レベル

回路図（スケマティック）エディタで，プリミティブ・セルを一つずつ置いて，つないで設計します．現在，ゲート・レベルで設計を始めることはほとんどありません．

しかし，ゲート・レベルの設計データは広く使われています． EDAツールは，ESLやRTLで記述したコードからゲート・レベルのネットリストを生成しているためです．そこで，ネットリスト・レベルとも呼ばれます

図4
同期設計と非同期設計
同期設計では，すべてのレジスタが，共通のクロック信号で動作する．

ネットリストのフォーマットとしては，Verilog HDLやVHDL，EDIF（electronic design interchange format）が使われています注2．

● 同期設計と非同期設計

最近の論理設計の基本は同期設計です（**図4**）．すべてのレジスタが，共通のクロック信号で動作します．

昔の論理設計は非同期設計でした．レジスタのクロック・ピンが，さまざまな内部信号に接続されるのです．非同期設計はタイミング検証がしずらく，設計生産性が上がらないなどの問題があったため，同期設計が基本と言われるようになりました．ただし，低消費電力や超高速動作が求められるLSIでは，非同期設計に注目することがあります．また，一つのLSIの中で複数のクロックを利用する場合，使用するクロックが異なるブロック間が非同期になることがあります．

● 同期リセットと非同期リセット

同期設計・非同期設計と似たものとして，同期リセットと非同期リセットがあります（**図5**）．

同期リセットは，動作中に何度もかかるリセットに適しており，非同期リセットは，起動時にかかるパワーON リセットに適しています．

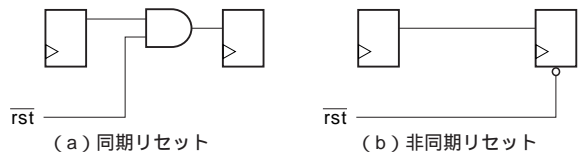



図5　同期リセットと非同期リセット
同期リセットは，動作中に何度もかかるリセットに適しており，非同期リセットは，起動時にかかるパワーON リセットに適する．

3　機能検証

機能検証とは，仕様を基にして記述した回路が，仕様通りに動作するかを確認する作業です．

意外と思われる方もいるでしょうが，機能検証は機能設計の2倍くらいの時間がかかると言われてます．このため近年，この機能検証が注目されてます．

LSIの大規模化や複雑化に伴い，工数が爆発的に増加し，不具合も抑えられなくなってきているためです．リスピン（設計不具合によるマスク再設計）の9割以上が，機能検証のもれに起因するというデータもあります．リスピンがおきると，ECO（engineering change order）と呼ばれる回路修正を行う必要が出てきます．

● 論理シミュレーション

機能検証は，主に論理シミュレーションという手法により，論理シミュレータというツールを使って行います．これは，ソフトウェア上で，回路の動作を模擬するものです．

入力信号に対して仕様に従って動作をするような入力パターンを与え，出力信号や内部ノードの状態を確認します．シミュレータからの出力は，テキスト形式のログやGUI（graphical user interface）による波形表示があります．

● アサーション

アサーションは，出力（観測）側の改善手法です．期待する動作をアサーション言語として埋め込み，シミュレーション中に違反動作をしたら，エラーを出力します．ABV（assertion baesd verification）と呼ばれることもあります．

アサーション言語としては，SVA（SystemVerilog Assertion）とPSL（Property Specification Language）があります．

● 制約付きランダム・テストベンチ

入力（制御）側の改善手法です．ユーザが付けた制約に従ったランダム・パターンを入力することで，コーナ・

注2：今後EDIFは使われなくなる可能性がある．Synopsys社のDesign Compiler2007.03ではEDIFはサポートされなくなる．

ケースを確実に検証します.

テストベンチ言語としては，SVTB（SystemVerilog Testbench）とe言語があります.

VMM（Verification Methodology Manual）も，SVTBによる制約付きランダムがベースの検証手法論です.

4 論理合成

論理合成とは，RTLで記述したコードをゲート・レベルのネットリストへ変換する処理です．論理最適化とテクノロジ・マッピングを行います.

● 論理最適化

論理最適化では，制御系の論理圧縮・展開と演算系のアーキテクチャ選択を行います.

最近では，配置ツールのタイミング収束能力が強くなったことから，論理合成は単にネットリストを作ってくれさえすればいい（品質は気にしない）と軽んじる人もいるようです．しかし，ゲート・レベルでは論理圧縮・展開しかできません．すなわち演算系のアーキテクチャ選択は，論理合成でしかできないのです.

乗算器を例に取ると，論理合成で低速なcsa（carry save array）が選択されてしまったら，後から変更できません．論理合成では，与えられた制約に従って，そのタイミングを満足させるために，適切なアーキテクチャを選択する必要があります.

● テクノロジ・マッピング

ゲート・レベルのネットリストを，半導体製造技術（単に「テクノロジ」と呼ぶことが多い）に応じたライブラリに割り当てます.

論理合成の出現により，一つのRTLからさまざまなテクノロジのネットリストを作成可能になりました.

● 制約

論理合成時に重要なのは，RTLのソース・コードと同時に入力する制約条件です．論理合成ツールは，この制約に従って処理を行います．設計規則制約（最大容量，最大ファンアウト，最大遷移時間），タイミング，面積などの制約を与えます.

タイミング制約としては（論理合成に限らず），SDC（Synopsys Design Constraint）というフォーマットが広く使われています.

● ゲート遅延と配線遅延

既存の論理合成は，限界に近づいてます．製造プロセスが90nm，65nmと微細化するにつれて，信号がゲートを通るときに生じる遅延時間よりも，配線を通るときに生じる遅延時間が支配的になってきているためです.

論理合成では配置配線が考慮されません．このため，配線遅延モデル（wire load model）による見積もりしかできません．ゲート遅延が支配的だった時代は，この見積もりに近い結果が得られていました．しかし，配線遅延が支配的になった昨今，見積もりから大きく外れることが問題となっています.

これを解決するのが，論理合成の中で仮配置を行い，配線容量を見積もる機能の出現です[注3]．ただしこの技術も，実際に配置で使うツールとEDAツール・ベンダが異なる場合に，データの整合性が問題になります．また，より正確さを求めようとしたときにはフロアプラン情報まで必要になるため，発展途上の技術です.

● パワー合成とゲーテッド・クロック

最近は，論理合成と同時にパワー合成を行えます.

今日のLSIでは，クロック信号による電力消費の割合が上がっています．同期式回路では，常にクロックが供給されるためです.

大規模LSIでは，すべての回路が常に動作しているわけではありません．そこで，動作しないで良い回路に対しては，クロックの供給を止めることで，消費電力を低減できます．このような回路構造を，ゲーテッド・クロック（gated clock），または「クロック・ゲーティング（clock gating）」と言います.

そこで，論理合成時に回路の動作を解析し，フリップフロップのデータ更新の条件に応じて，データ・ラインのマルチプレクサをクロック・ラインのANDゲートに自動変換します（**図6**）.

ゲーテッド・クロックでは，クロック・ラインにはCGC（clock gating cell）を使う場合がほとんどです.

注3：Synopsys社のTopographical TechnologyやCadence社のPLE（Physical Layout Estimator）がある.

図6　ゲーテッド・クロック
動作しないでよい回路に対してはクロックの供給を止める．

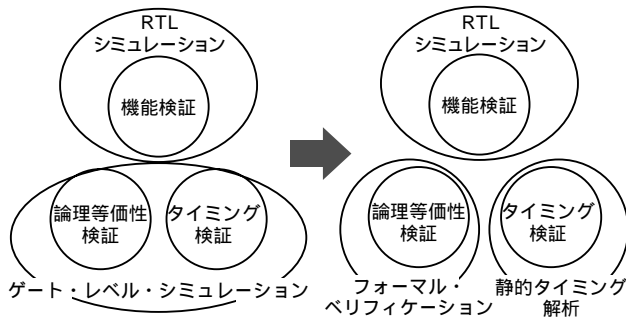
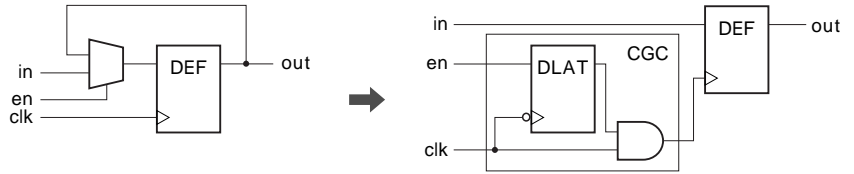



図7　検証の役割
ゲート・レベルでは，タイミング検証と論理等価性検証を行う．

図8　セル遅延と配線遅延
タイミング検証も行う場合は，タイミング情報としてSセル遅延と配線遅延が含まれるファイルを用意する必要がある．

5　タイミング検証と論理等価性検証

ゲート・レベルでは，タイミング検証と論理等価性検証を行います（**図7**）．

● セル遅延と配線遅延

タイミング検証を行う場合も，タイミング情報としてSDF（standard delay format）と呼ばれるセル遅延と配線遅延が含まれるファイルを用意する必要があります（**図8**）．

● ゲート・シミュレーション―ダイナミック手法

ゲート・シミュレーションでは，機能検証で使用したパターンを流し込んで，論理シミュレーションの結果との比較を行います．一致していれば論理的に等価であり，かつタイミングも満足していると間接的に判断できます．パターンを用いるので，ダイナミック（動的）手法と呼びます．

この手法の限界は，どこまで検証できているかが，入力パターンに依存するということです．論理的に等価でない部分やクリティカル・パス（一番タイミングが厳しいパス）が，入力したパターンによって動作しなければ，発見できません．さらに，タイミング検証においては，ばらつきを考慮できないことが問題になっています．

このため，最近ではダイナミック手法からスタティック（静的）手法に移行してます．

● スタティック手法

スタティック手法は，論理等価性検証やSTA（static timing analysis）を用いる手法です．スタティック手法では，パターンが不要で処理時間が短い，100%漏れなく検証できるなどの特徴があります．

ダイナミック手法とスタティック手法のどちらをどのように採用するかは，サインオフ条件次第です．サインオフ条件とは，製造側が発注側に求める検証手法や検証ツールについての条件になります．

● フォワード・アノテーションとバック・アノテーション

タイミング検証は，レイアウト前に仮配線遅延のSDFで行うことがあります．これを「フォワード・アノテーション」と言います．また，レイアウト後に実遅延のSDFで行うことを「バック・アノテーション」と言います．

バック・アノテーションは必須ですが，フォワード・アノテーションは省略したり，論理合成で簡易的に行うことも多いようです．

● 論理等価性検証

論理等価性検証は「形式的検証」や「フォーマル・ベリフィケーション」と呼ばれることもあります[注4]．

注4：形式的検証やフォーマル・ベリフィケーションには，論理等価性検証とプロパティ・チェックの2種類がある．

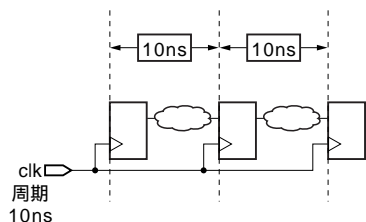



図9　STA
クロックを定義して周期を与える．

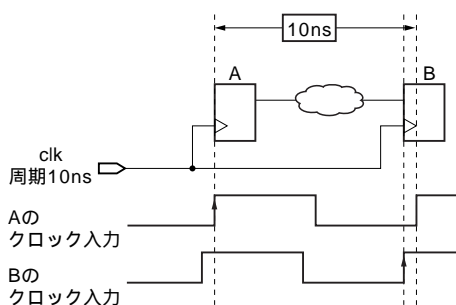
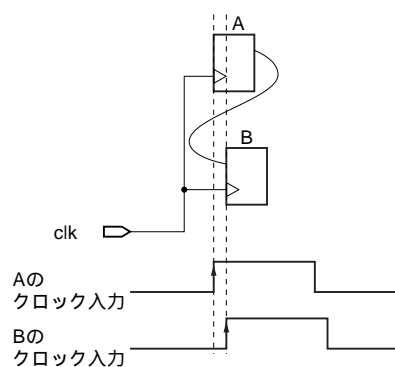



図10　セットアップ検証とホールド検証
セットアップ検証は，二つのクロック・エッジ間の検証．ホールド検証は，同一のクロック・エッジにおける検証．

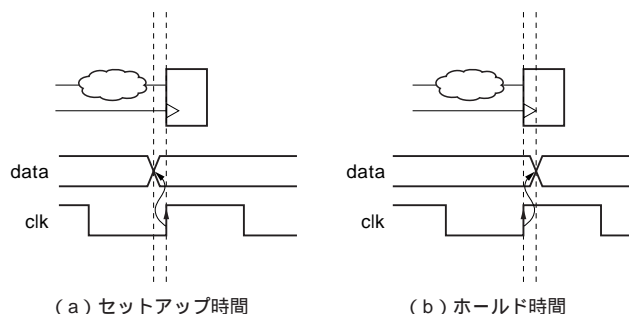



図11　セットアップ時間とホールド時間
セットアップ時間は，クロック・エッジよりも前にデータが確定しておく必要がある時間．ホールド時間は，クロック・エッジよりも後にデータを保持しておく必要がある時間．

これは，二つの設計が論理的に等価であるかどうかを静的に検証する手法です．設計データに何らかの変更があった場合に実施します．例えば，設計ツールの入力と出力に対して行います．機能検証を行ったRTLに対して，設計ツールによりRTLのコードから生成されたゲート・レベルの回路が，RTLの論理を継承してるかどうかを確認します．ゲート・レベル同士で検証することもあります．

● STA（static timing analysis）

ネットリストとタイミング情報（SDF）に対して，タイミング制約（SDC）を与えることで，タイミングを満足しているかどうかを確認します．

STAの鍵はSDCです．与えられた制約についてしか確認しないためです．漏れがあると，パターン依存のあるゲート・シミュレーションで発見できるようなタイミング・エラーでさえも見逃してしまいます．

同期回路であれば，**図9**のようにクロックを定義して周期を与えるだけで，すべてのレジスタ間にその周期の制約が与えられ，容易に確認が可能です．

(1) セットアップ検証とホールド検証

STAで重要になるのは，セットアップ検証とホールド検証です（**図10**）．

セットアップ検証は，二つのクロック・エッジ間の検証です．レジスタ間の信号は，制約条件で与えられた最大遅延時間以内に到達しなければなりません．

ホールド検証は，同一のクロック・エッジにおける検証です．レジスタ間の信号は，最小遅延時間が0以上で到達しなければなりません．

(2) セットアップ時間とホールド時間

実際の回路では，セットアップ時間とホールド時間を考慮しなければなりません（**図11**）．

セットアップ時間は，クロック・エッジよりも前にデータを確定しておく必要がある時間です．

ホールド時間は，クロック・エッジよりも後にデータを保持しておく必要がある時間です．

(3) マルチサイクル・パスとフォルス・パス

クロック以外で重要な制約として，タイミング例外（timing exception）があります．

周期の複数倍で到達すればよい「マルチサイクル・パス」や，実際の動作では問題ないので無視してよい「フォルス・パス」を指定します．

6　レイアウト（配置配線）

レイアウト（または配置配線；P&R ： place & route）

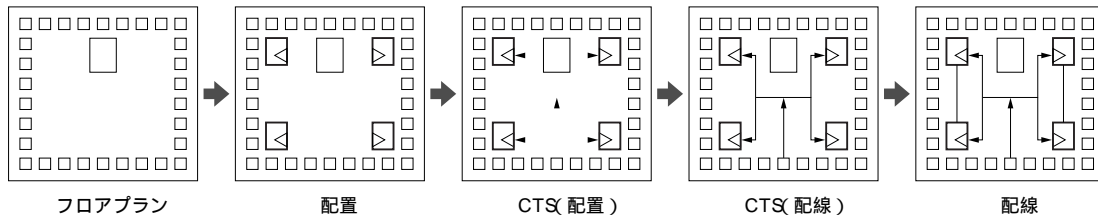



図12　レイアウト(配置配線)
論理情報のネットリストを基に，フロアプランと配置，クロック・ツリー合成，配線を行い，半導体としての物理的な設計データに変換する工程．

は，論理情報のネットリストを基に，フロアプランと配置，クロック・ツリー合成，配線を行い，半導体としての物理的な設計データに変換する工程です(**図12**)．半導体を製造するために用いるレイアウト・フォーマットであるGDS IIを生成します．

レイアウトの使命は，

- 配置配線を物理的に行えること
- タイミングが仕様通りに収束すること

の2点になります．

図13　クロック・ツリー合成
クロック・スキューをできるだけ抑え，理想的な回路に近づける．

● フロアプラン

フロアプランでは，I/OセルやPLL(phase-locke loop)，RAM(random access memory)といったハード・マクロを配置します．次に，すべてのスタンダード・セルを配置します．そして，レジスタの配置が決定したところで，これらのクロック遅延を合わせるために，クロック・ツリー合成(CTS：clock tree synthesis)を行います．最後にセル間を配線します．

● クロック・ツリー合成(CTS：clock tree synthesis)

同期設計はクロック・ラインが命です．なぜなら，すべてのレジスタのクロック・ピンに対して，まったく同じタイミングでクロックが入力されていると仮定して，設計するからです．

しかし，実際のチップでは，配線を通る際に遅延時間が生じます．そして，物理的な配置配線によってレジスタごとにわずかな遅延時間の差分が生じます．これを，「クロック・スキュー」と言います．クロック・スキューをできるだけ抑え，理想的な回路に近づけるのがクロック・ツリー合成です(**図13**)．

● タイミング・ドリブン・レイアウト(TDL：timing driven layout)

一般的にレイアウト工程では，各セルの接続性に従って配置配線を行います．この際，タイミングはまったく考慮されません．

しかし，タイミングを考える際には，配置が肝と言えます．なぜなら，物理的に離れているものは，どんなに頑張っても配線距離が長くなり，遅延時間が大きくなるからです．

タイミング・ドリブン・レイアウトとは，タイミング制約(SDC)を与えることで，タイミングを考慮しながら配置配線を行います(**図14**)．すなわち，タイミング制約が厳しいセル同士を近づけて配置配線することで，タイミング的に良いレイアウトを実現することが可能になります．タイミングが厳しい設計には，非常に有効な手法で，一般的に使われています．

とはいえ，これでタイミング・エラーをまったくなくせるわけではありません．残るタイミング・エラーを解決するのはエンジニアです．よくも悪くも，ツールに人間が勝つ(ツールが人間に負ける)ところと言えましょうか．

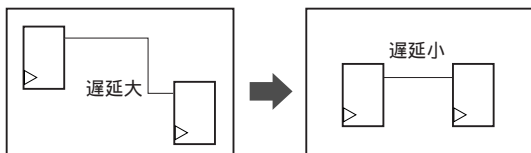



図14　タイミング・ドリブン・レイアウト
タイミング制約が厳しいセル同士を近づけて配置配線する．

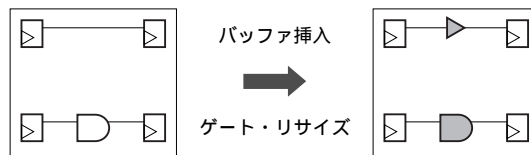



図15　タイミング収束
リピータ（バッファ）挿入やゲート・リサイズなどの最適化を行う．

● タイミング収束

タイミング・エラーを解決する作業を，タイミング収束（クロージャ）と呼びます．配置から考え直すのであれば，強制配置やグルーピングを行います．配置を変えないのであれば，最適化を行うことで解決します．タイミング・ドリブン・レイアウト・ツールの中には，配置と同時に，最適化を行ってくれるものがあります．

最適化には，リピータ（バッファ）挿入やゲート・リサイズがあります（**図15**）．また，論理合成のように論理圧縮・展開まで行ってくれるものもあります．

7　レイアウト検証と製造

レイアウト検証には，LVS（layout versus schecmatic）とDRC（design rule check）の二つがあります．

● LVS（layout versus schecmatic）とDRC（design rule check）

LVSは，ネットリストとGDS IIとを比べて，素子や素子間が完全に同一であるかどうかを検証します．

DRCでは，製造プロセスに合わせて幾何学的な設計ルールのチェックを行います．具体的には，配線幅や配線間隔などが，プロセスのルールに合致しているか否かです．

● OPC（optical proximity correction）

近年，GDS IIがそのままマスクになることはありません．微細化に伴い，OPC（optical proximity correction；光近接効果補正）による補正が必要になっています．

これは露光時の回折現象などを考慮して，補正パターンを追加するなどの処理です．OPCを行ったものがマスク描画用のEB（electron beam）データになります．

● DFM（design for manufacturing）

近年，DFM（design for manufacturing）という言葉をとく耳にします．製造プロセスの微細化に伴い，チップ製造が困難になっているためです．

現在は，露光で問題になりそうなポイントを，露光シミュレーションで発見して補正したり，OPCによるさらなる補正を指すことが多いようです．

ウェハの歩留り（良品率）を上げることを目標とする技術は，DFY（design for yield）と呼ばれることもあります．新しい技術のため，範囲はあいまいです．

今後は，より上流に上ってきて，レイアウト工程で補正するような技術が期待されています．

8　テスト容易化設計

テスト容易化設計（DFT ：design for testability）は，スキャンを意味する場合が多いようです．スキャンとは，テスト回路を自動挿入するスキャン回路挿入（スキャン・インサーション）とATPG（automatic test pattern generation）の組み合わせで実現します．

● スキャン回路挿入（スキャン・インサーション）

スキャン・インサーションで挿入されるテスト回路は，全レジスタをつないだ巨大なシフト・レジスタです．これをスキャン・チェーンと言います．任意のレジスタに対して，値を書いたり読んだりすることが可能になります．

まず，全レジスタをスキャン・セルと呼ばれるマルチプレクサ付きレジスタに置き換えます．マルチプレクサの片側は，これまでのデータ入力に接続します．もう片側を使ってスキャン・チェーンを構築します（**図16**）．

現在は，論理合成の段階でいきなりスキャン・セルを用いることが一般的です．ただし，スキャン・セルを使うとマルチプレクサ1段分だけ面積のオーバヘッドがあることに注意が必要です．

スキャン・チェーンの入口をスキャン・イン，出口をスキャン・アウトと呼びます．通常の回路とテスト回路を切

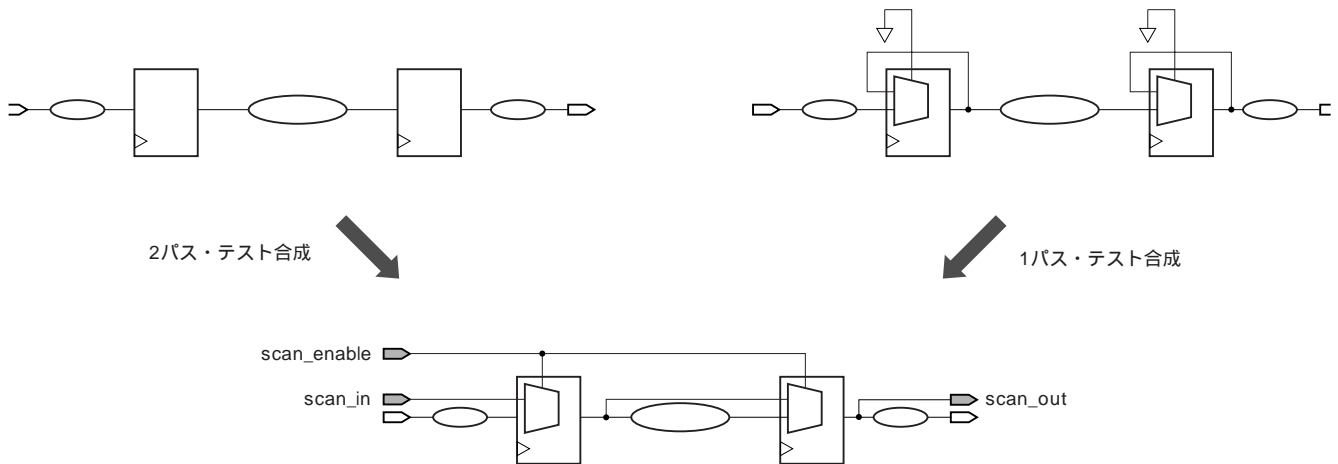



図16　スキャン回路挿入(スキャン・インサーション)
全レジスタをスキャン・セルと呼ばれるマルチプレクサ付きレジスタに置き換える．マルチプレクサの片側はこれまでのデータ入力に接続する．もう片側を使ってスキャン・チェーンを構築する．通常のレジスタとして合成された回路からレジスタを置き換える2パス・テスト合成と，あらかじめマルチプレクサ付きレジスタを合成する1パス・テスト合成がある．

り替えるのがスキャン・イネーブルです．スキャン・イネーブルが有効のときをシフト，無効のときをキャプチャと言います．シフトで任意の値を設定し，キャプチャで通常回路を伝播した値を取り込み，再度シフトで外部へ出力します．

● ATPG(automatic test pattern generation)

半導体の不具合には，バグと故障があります．

バグは，機能が仕様通りに動作しない不具合です．バグがあれば製造したすべてのチップで発生します．これは，機能検証で検出すべきものです．

一方，故障は製造上の不具合です．多くの場合，一部のチップでのみ発生します．これは，LSIの出荷テストで検出すべきものです．

ATPG(automatic test pattern generation)とは，出荷検査時のLSIテスタ(ATE：automatic test equipment)で使用するテスト・パターンを自動的に生成することです．ほとんどは，スキャン回路を用いたパターンの生成を指します．

LSIテスタでは，テスト・パターンを流すためには，信号の周期・波形などを定義して制御するテスト・プログラムも必要になります．

● 圧縮パターン・テスト

LSIの大規模化に伴い，テスト・パターンが膨大になり，全コストに占めるテスト・コストの割合が上がっています．テスト・パターンが長くなるとテスト時間がかかるためです．この問題を解決するため，圧縮パターン・テストが広く用いられています(**図17**)．

スキャン・チェーンを分割することにより，一つのスキャン・チェーンを短くします．これにより，全レジスタへの読み出しや書き込みの時間を短縮できます．

スキャン・チェーンの入口には伸張回路が，出口には圧縮回路が付加されます．これにより，スキャン・テストに必要な信号数は変わりません．そのため，テスタが持つデータ量も，同様に小さくできます．

● BIST(built in self test)

テスタからパターンを与えるのではなく，パターン発生器と出力圧縮器をLSIに内蔵するテスト手法です．これにより，テスタからはごく一部の制御パターンを与えるだけですむようになります．

論理回路に適用するロジックBISTよりも，メモリ(RAM)に適用するRAMBISTが一般的です．RAMでは，印加すべきパターン・アルゴリズムが明確になっているためです．

論理回路の場合は，パターン発生器から与えるのはランダム・パターンになります．従って，故障検出率は上がりにくくなります．そこで，PLLを活用して大量のパターンでも高速に走らせる工夫を行います．

圧縮パターン・テストとBISTの違いを**表2**に示します．半導体の良否だけでよければ圧縮パターン・テスト，機器

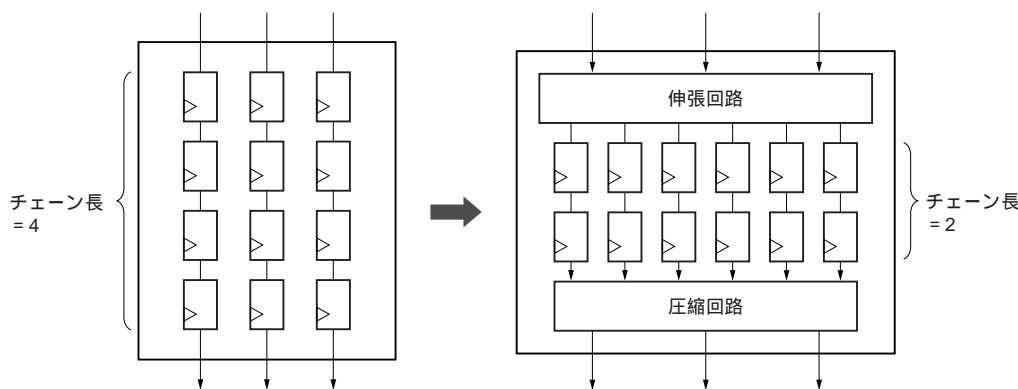



図17
圧縮パターン・テスト
スキャン・チェーンを分割することにより，一つのスキャン・チェーンを短くする．

表2　圧縮パターン・テストとBISTの違い

	圧縮パターン・テスト	BIST
テスト回路の実装	容易	困難
面積オーバヘッド	小	大
パターン生成	ATPG	ランダム
パターン数	大量ではない	大量
PLLクロック使用注	不可能	可能
圧縮率	高くない	高い
検出率	高い	高くない
オンボード・テスト	不可能	可能

注：シフト時（キャプチャ時ではない）

メーカからの要求により，オンボード・テストも必須となっているのであればロジックBISTとなりそうです．

9　消費電力

半導体製造プロセスの微細化やLSIの大規模化に伴い，消費電力にかかわる話題が増えています．

● パワー検証（消費電力見積もり）

LSIの消費電力は，スイッチング動作に伴う電力が支配的です．特に同期回路では，クロックに合わせて常にチャージとディスチャージが繰り返されます．

パワー検証においては，どのネットがどのくらい遷移するかが問題になります．これには，遷移確率を一律に与える手法と，シミュレーションから各ネットの遷移情報を得る手法があります．前者はおおざっぱな見積りでしかありません．後者は比較的正確に見積もることが可能になります．しかし，LSI大規模化に伴い，実動作に近いパターンを与えることが困難になってきています．

実際には，シリーズ品などの相対比較には使えるようですが，消費電力の絶対値についてはまだ誤差が大きいようです．

● リーク電流

製造プロセスの微細化に伴って，リーク電力が占める割合が増大してきてます．リーク電力とは，クロックを止めた状態でも，流れている漏れ電流のことです．

LSI全体のリーク電流を抑える方法として，マルチV_tのライブラリがあります．マルチV_tとは，しきい値電圧V_tが低いセルと高いセルを用意することです．速度が要求されるところだけにV_tが低いセルを使って，ほかをV_tが高いセルを使うことで，リーク電力を抑えようというものです．

低消費電力を実現する手法としては，ほかにレベル・シフタを使う多電源回路や，アイソレーション・セルなどを使う電源遮断回路などがあります．

● CPF（Common Power Format）とUPF（Unified Power Format）

最近，消費電力の仕様記述言語を標準化しようという動きがあります．PFI（Power Forward Initiative）が作成したCPF（Common Power Format）と，UPF-TS（Unified Power Format-Technical Subcommittee）が作成したUPF（Unified Power Format）です．

いつの時代も標準化の前は，2強が争うのは世の常でしょうか．

ふるかわ・ひろし
NECマイクロシステム（株）
hiroshi.furukawa@nms.necel.com